**Final Project**

**Quadruped Robotics**

**Exploration of Reinforcement Learning techniques in Quadruped Robotics Environment**

**Reinforcement learning**

**Academic Year 2024-2025**

**Student Information**

|  |  |  |
| --- | --- | --- |
| Student names | Email addresses | Description |
| HEEHYEONG Nam | heehyeong.nam@opendeusto.es | Conducted experiments with the PPO algorithm, incorporating Optuna for hyperparameter optimization, modifying environmental setups, and performing TensorBoard analysis to evaluate performance. |
| BOUKSAIM Wissal | wissal.bouksaim@ opendeusto.es | Performed a series of experiments with the SAC algorithm, integrating Optuna for hyperparameter tuning and applying the best `forward\_reward\_weight` value identified during the PPO experiments. Furthermore, I utilized TensorBoard to monitor, analyze, and present the model's performance, and I contributed to a part of the project presentation. |
| EL KACHACH Manal | Manal.elkachach@opendeusto.es | Worked on curriculum learning strategy, Optuna tuning, report, and presentation. |

**Contents**

1. Introduction
2. Environment Setup and Installation
3. Experiments and Results
   1. PPO Algorithm
   2. SAC Algorithm
   3. Curriculum Learning
4. Comparison and Analysis
5. Conclusion and Insights
6. Future Directions
7. **Introduction**

In this project, we implemented and explored reinforcement learning (RL) techniques on the Quadruped Robotics environment. The environment was chosen due to its realistic challenges and relevance to robotic control tasks.

The key objectives of the project were:

* Understanding and applying RL algorithms such as Proximal Policy Optimization (PPO) and SAC.
* Experimenting with reward modification and hyperparameter tuning.
* Incorporating curriculum learning and automated hyperparameter optimization (Optuna).

1. **Environment Setup and Installation**

The environment was set up following the official documentation from Gymnasium:  
**Source**: <https://gymnasium.farama.org/main/tutorials/gymnasium_basics/load_quadruped_model/>

1. **Download the whole MuJoCo Menagerie Collection:**

git clone <https://github.com/google-deepmind/mujoco_menagerie.git>

1. **Install the essential libraries:**

* pip install mujoco
* pip install “gymnasium>=1.0.0”
* pip install stable\_baselines3

1. **Implement the Quadruped Robotics environment:**

import gymnasium

def create\_env():

    # Create and return the default environment

    env = gymnasium.make(

        'Ant-v5',

        xml\_file='./mujoco\_menagerie/unitree\_go1/scene.xml',

        forward\_reward\_weight=1,

        ctrl\_cost\_weight=0.05,

        contact\_cost\_weight=5e-4,

        healthy\_reward=1,

        main\_body=1,

        healthy\_z\_range=(0.195, 0.75),

        reset\_noise\_scale=0.1,  # reset\_noise\_scale=0.0 for a deterministic environment

        frame\_skip=25,

        max\_episode\_steps=1000,

        render\_mode='human',

    )

    return env

# Test code

if \_\_name\_\_ == "\_\_main\_\_":

    env = create\_env()

    # Reset the environment

    obs, info = env.reset()

    done = False

    # Run a test episode

    while not done:

        action = env.action\_space.sample()

        obs, reward, done, truncated, info = env.step(action)

        print(f"Reward: {reward}")

        # Render the environment (if a viewer is available)

        try:

            env.render()

        except Exception as e:

            print(f"Render error: {e}")

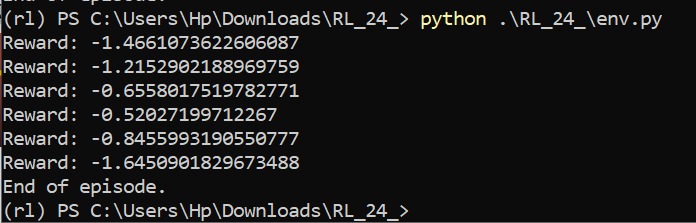
        if done or truncated:

            print("End of episode.")

            break

    env.close()

1. **Verified the environment functionality: (see Video\_test)**

The test environment ran successfully, producing a series of rewards. The negative rewards indicate initial random actions, which is typical for untrained agents.

1. **Resolution and Testing**
2. **PPO Algorithm:**

We employed the **Proximal Policy Optimization (PPO)** algorithm as the initial strategy for training our agent in the Quadruped Robotics environment. To identify the optimal hyperparameters for PPO, we utilized **Optuna** for automated hyperparameter tuning.

* **Hyperparameter Tuning with Optuna :**

We conducted **60 trials** of hyperparameter optimization. The optimization history plot below illustrates the progress across trials, where the objective value improved significantly over time as better hyperparameter combinations were discovered.

텍스트, 스크린샷, 라인, 도표이(가) 표시된 사진

자동 생성된 설명

Best Hyperparameters Identified :

* + - * **learning\_rate**: 0.000974
      * **gamma**: 0.9338
      * **n\_steps**: 1024
      * **ent\_coef**: 1.966e-5
      * **batch\_size**: 160
* **Environmental Reward Modification:**

Using these hyperparameters, we trained our agent to evaluate their performance.

In addition, we tried to consider the impact of environmental reward settings, particularly focusing on the forward rewards. Since maximizing forward movement is crucial for efficient training, we increased the “forward\_reward\_weight” from 1 to 1.5 to identify the most suitable reward configuration.

1. “forward\_reward\_weight” =1(default reward)

텍스트, 메뉴, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. “forward\_reward\_weight” =1.5

텍스트, 메뉴, 스크린샷, 폰트이(가) 표시된 사진

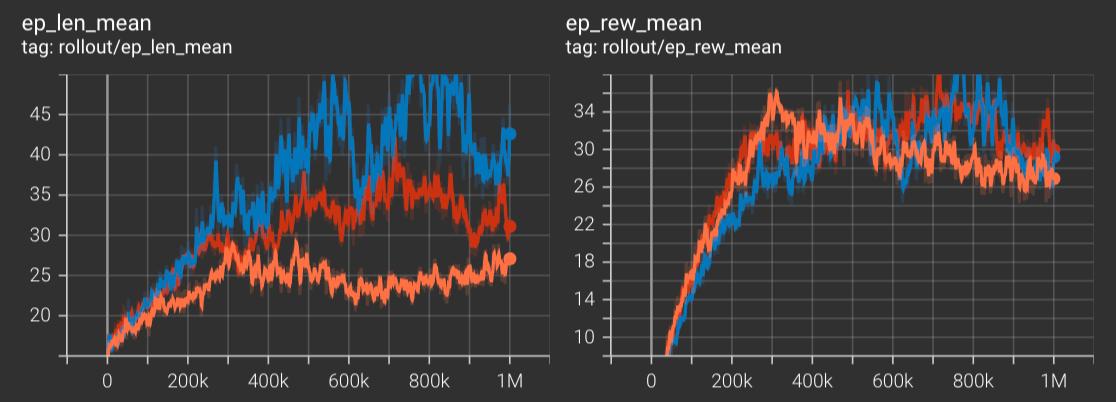
자동 생성된 설명

Increasing forward\_reward\_weight to **1.5** resulted in an improvement of **1.5** in ep\_rew\_mean but may caused instability, as the agent prioritized moving forward excessively while neglecting balance and control.

Therefore, we concluded that keeping forward\_reward\_weight at its default value of 1 is the optimal setting for stable training with PPO.

* **Training Results:**

We trained the PPO agent using Optuna. The training process was monitored using **TensorBoard** for performance analysis.



\*Blue: frw=1, Red: frw=1.2, Orange: frw=1.5

1. **Mean Episode Length (eval/mean\_ep\_length):**

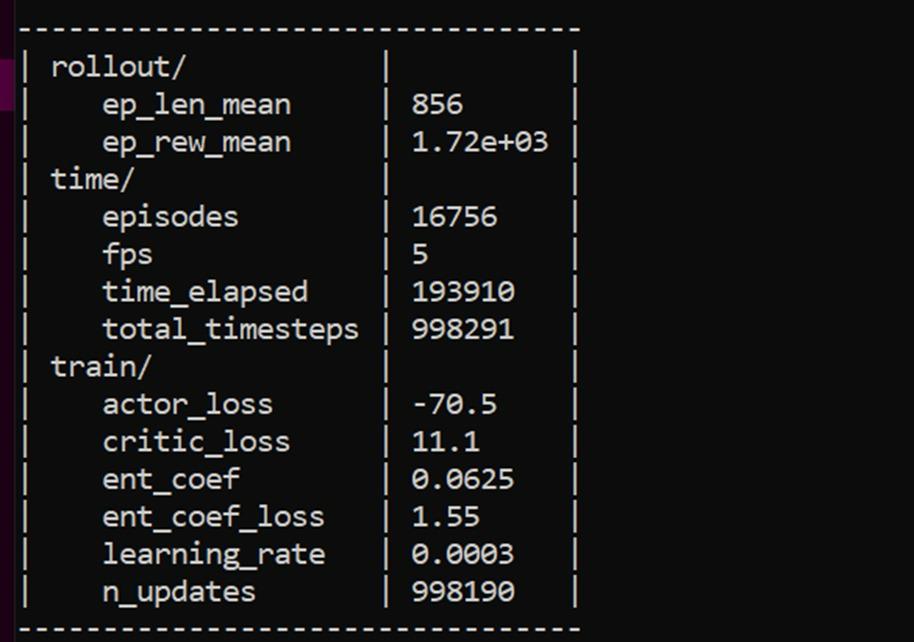
* **Observation**:
* The mean episode length begins with increasing trend around 20 steps, but it showed occasionally high variability
* Over time, the episode length stabilized, peaking around **45–50 steps** in the blue experiment. By the end of training (at ~1 million steps), the values converge to a relatively stable range.
* The red curve shows more steady behavior with less fluctuation compared to the blue curve, remaining slightly lower overall.
* **Interpretation**:
* Early fluctuations indicate that the agent is actively exploring the environment, attempting a wide range of actions to discover effective strategies.
* Variations in the curves (e.g., blue showing higher episode lengths) reflects differences in experimental setups(in this case, forward\_weight\_reward) is affecting exploration and learning rates.

1. **Mean Reward (eval/mean\_reward):**

* **Observation**:
* At the beginning, the mean reward is low, reflecting random exploration by the agent. The rewards gradually increase over the first 200,000 steps across all experiments.
* The blue and red curves peak at **around 34**, showing steady improvement. However, after 400,000 steps, the rewards in both curves fluctuate, with some decline observed around 800,000–1 million steps.
* Overall, the final rewards across experiments remain consistent in the range of **28–34**, suggesting a training process with some instability.
* **Interpretation**:
* The later fluctuations and slight declines in rewards may indicate challenges in balancing exploration and exploitation or overfitting to specific strategies that are less robust.
* Entropy regularization in the learning process likely contributed to robust exploration early on, but fine-tuning may be required to reduce instability at later stages.

1. **SAC Algorithm:**

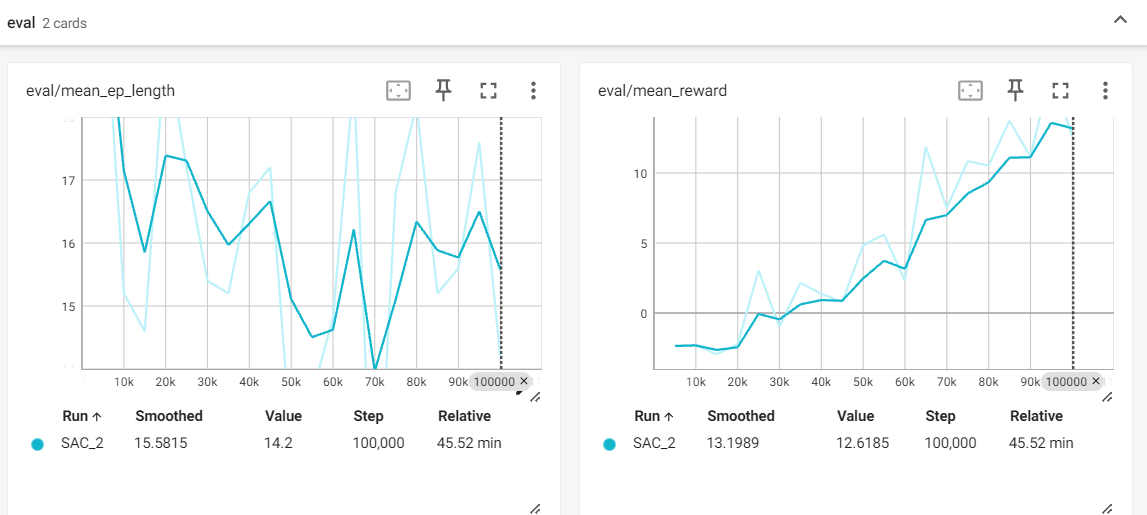
We trained our agent using **Soft Actor-Critic (SAC)** due to its superior performance in handling continuous action spaces and entropy-based exploration.

****

Initial results showed a **mean reward of 1723**, significantly outperforming PPO.(see VideoSAC)

* **Why SAC Outperformed PPO** :

SAC’s entropy-regularized objective encourages exploration, leading to better policy convergence. It can handle more complex reward functions and environments effectively.

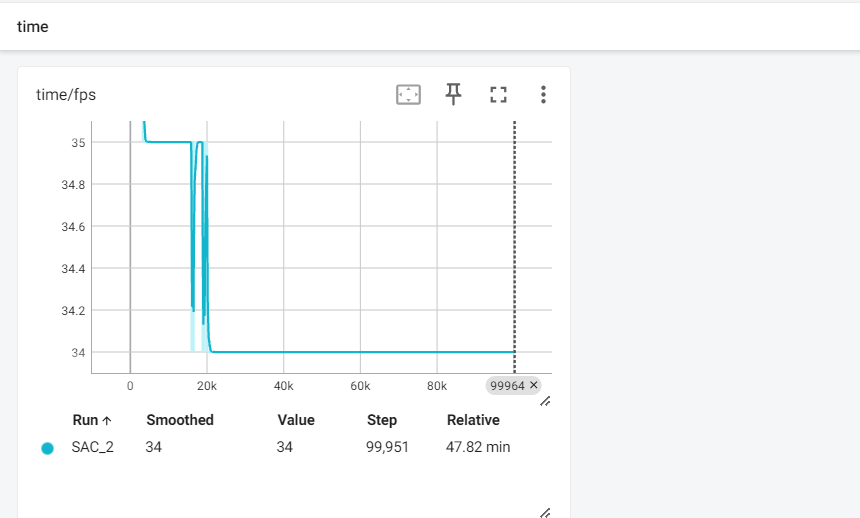
* **TensorBoard: **

1. **Mean Episode Length (eval/mean\_ep\_length):**

* **Observation**:
* At the beginning of training, the mean episode length fluctuates significantly, with episodes occasionally being longer (~17 steps).
* As training progresses, the episode length begins to stabilize, averaging around **14–15 steps** by the end of the training process.
* **Interpretation**:
* Early fluctuations indicate that the agent is still exploring the environment and learning to maintain balance.
* Stabilization of episode length reflects the agent's improved policy, as it has learned to operate efficiently and terminate episodes after achieving favorable conditions.
* Shorter, consistent episode lengths suggest that the agent has optimized its strategy for stability and reward maximization.

1. **Mean Reward (eval/mean\_reward):**

* **Observation**:
* Initially, the agent achieves near-zero rewards as it explores the action space randomly.
* Over time, the reward steadily increases, reaching a final value of approximately **12.6** at 100,000 steps.
* This upward trend is smooth and consistent, reflecting robust learning.
* **Interpretation**:
* SAC's entropy-regularized objective ensures exploration during the early phase, allowing the agent to avoid local optima.
* The steady improvement in mean reward demonstrates that the SAC algorithm effectively learns to maximize rewards while maintaining exploration.
* The gradual convergence highlights SAC’s stability in training, as the agent refines its policy incrementally.

****

1. **Frames Per Second (time/fps):**

* **Observation**:
* The FPS plot shows an initial fluctuation (around **35 FPS**) before stabilizing at approximately **34 FPS** for most of the training process.
* **Interpretation**:
* Stable FPS ensures efficient training and computational performance.
* Minor drops early in training are due to initialization overhead and environment setup but do not impact the overall process.
* The consistent FPS confirms that the SAC model was trained under optimal computational conditions without unnecessary slowdowns.

1. **Curriculum Learning Implementation:**

To further improve training stability and efficiency, **Curriculum Learning** was introduced. The core idea of Curriculum Learning is to gradually increase task difficulty as the agent progresses, making the learning process smoother.

* **Code Implementation** :

We implemented a **CurriculumCallback** to adjust the environment difficulty dynamically. Specifically:

**Curriculum Strategy** :

1. Start with **easy conditions** (wide height range).
2. Gradually reduce the range, making it harder for the agent to maintain balance.
3. Allow the agent to adapt incrementally to these tougher conditions.

from stable\_baselines3 import SAC

from stable\_baselines3.common.callbacks import CheckpointCallback, BaseCallback, CallbackList

from env import create\_env  # Import the environment from env.py

# Curriculum Learning Callback

class CurriculumCallback(BaseCallback):

    def \_\_init\_\_(self, verbose=0):

        super(CurriculumCallback, self).\_\_init\_\_(verbose)

    def \_on\_step(self) -> bool:

        if self.num\_timesteps % 100\_000 == 0:

            env = self.training\_env.envs[0]

            if hasattr(env, "healthy\_z\_range"):

                new\_range = max(env.healthy\_z\_range[0] - 0.01, 0.195)

                env.healthy\_z\_range = (new\_range, 0.75)

                print(f"Updated healthy\_z\_range to {env.healthy\_z\_range}")

            else:

                print("Warning: 'healthy\_z\_range' attribute not found in the environment.")

        return True

# Directory for saving checkpoints

checkpoint\_dir = './checkpoints\_sac/'

# Create the environment (without RecordVideo)

env = create\_env()

# Initialize the SAC model with custom hyperparameters

model = SAC(

    "MlpPolicy",

    env,

    learning\_rate=1e-4,

    batch\_size=512,

    gamma=0.98,

    verbose=1

)

# Checkpoint Callback

checkpoint\_callback = CheckpointCallback(

    save\_freq=20\_000,

    save\_path=checkpoint\_dir,

    name\_prefix="sac\_checkpoint"

)

# Curriculum Learning Callback

curriculum\_callback = CurriculumCallback()

# Combine Callbacks

callback = CallbackList([checkpoint\_callback, curriculum\_callback])

# Train the SAC agent with Curriculum Learning

model.learn(total\_timesteps=1\_000\_000, callback=callback)

# Save the final trained model

model.save("sac\_ant\_curriculum")

# Record a video of the trained agent

video\_dir = "./videos/"

eval\_env = create\_env()  # Create a new environment for evaluation

eval\_env = RecordVideo(eval\_env, video\_folder=video\_dir, episode\_trigger=lambda episode\_id: True)  # Record all episodes

# Reset the environment

obs, info = eval\_env.reset()

done = False

while not done:

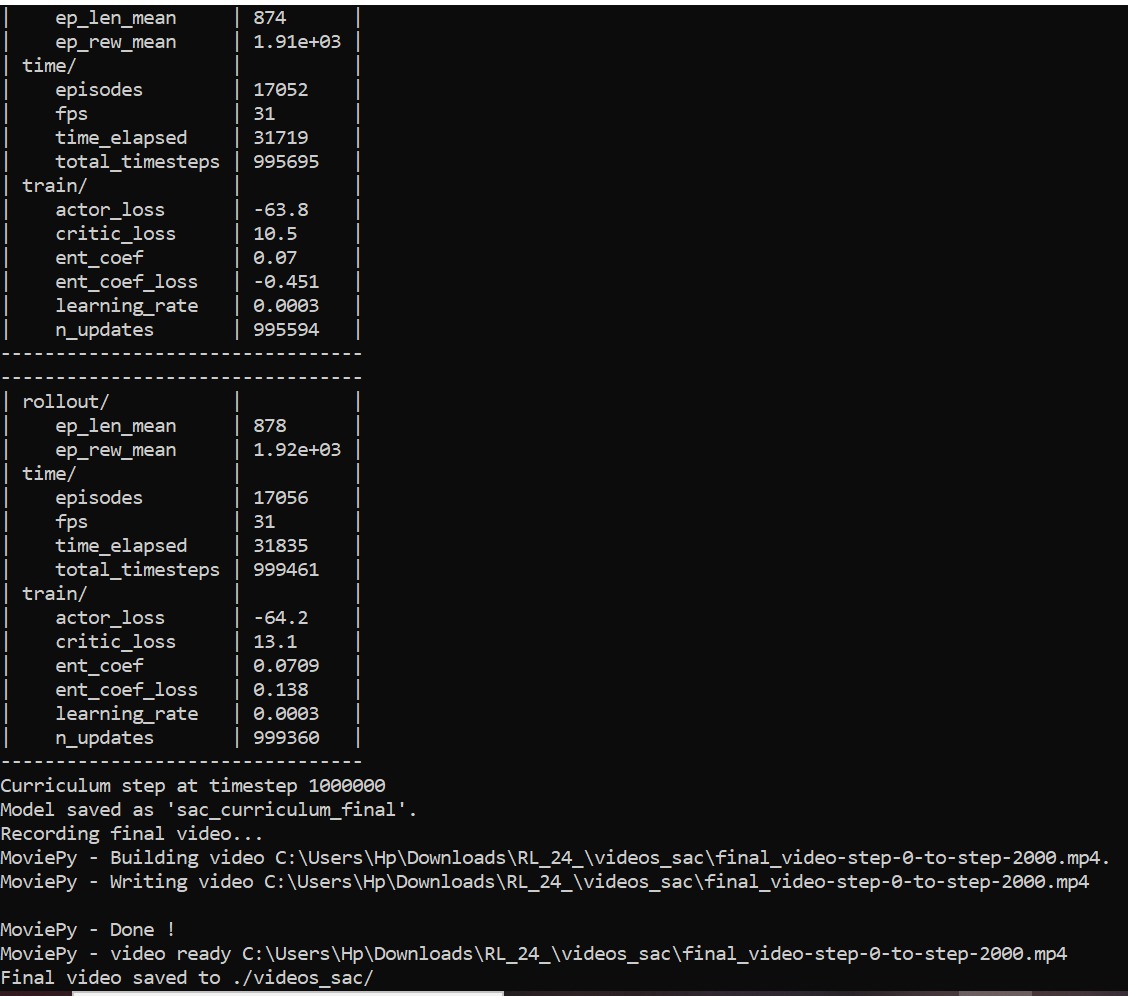
    action, \_ = model.predict(obs, deterministic=True)

    # Take a step in the environment

    obs, reward, terminated, truncated, info = eval\_env.step(action)

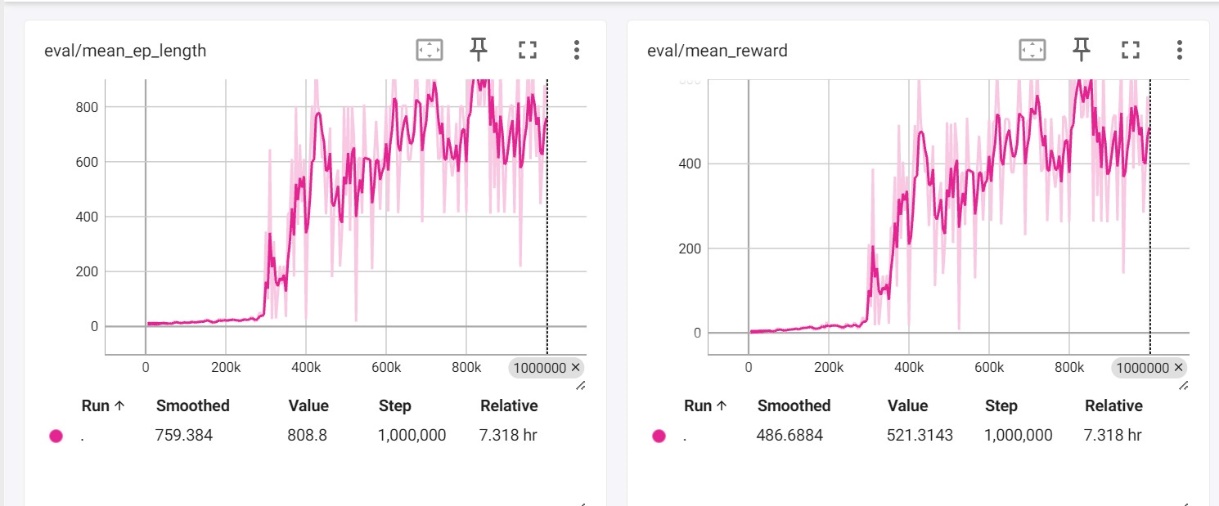
    done = terminated or truncated

eval\_env.close()

****

After implementing Curriculum Learning, the SAC agent achieved a new reward of **1913**, an improvement over the baseline of 1723. This confirms that Curriculum Learning helps the agent generalize better and adapt progressively to more difficult environments. (see Video 3)

* **TensorBoard:**

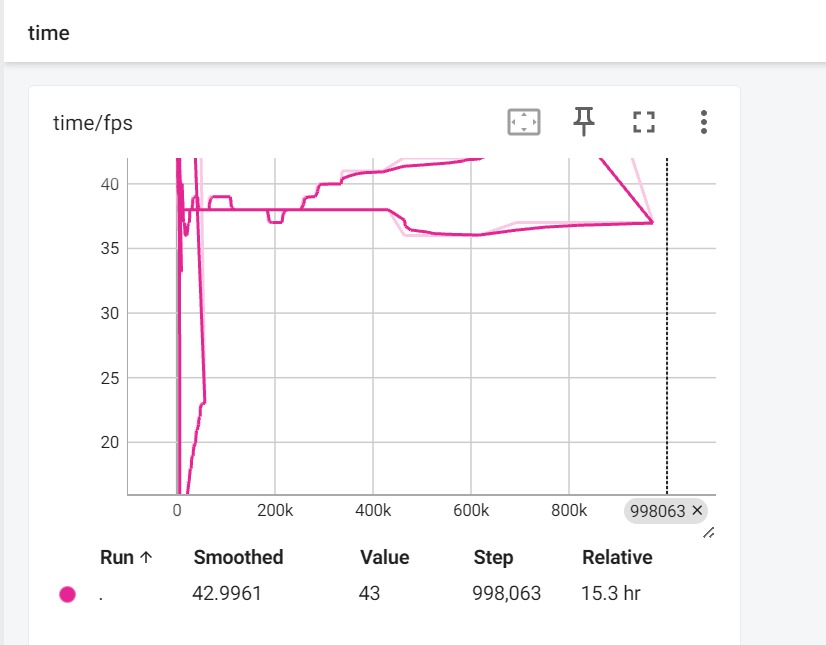
****

1. **Mean Episode Length (eval/mean\_ep\_length):**

* Episode length increased steadily, stabilizing at **808 steps**, reflecting the agent's ability to adapt to harder tasks over time.

1. **Mean Reward (eval/mean\_reward):**

* Rewards rose consistently, peaking at **521**, demonstrating improved agent performance with progressive difficulty.



1. **Frames Per Second (time/fps):**

FPS remained stable at **43**, with minor drops during environmental updates, ensuring efficient training.

1. **Hyperparameter Tuning with Optuna:**

To achieve optimal performance, we employed **Optuna** to tune SAC hyperparameters. The following hyperparameters were selected for optimization:

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- |   **Hyperparameter** | **Search Space** | **Reason for Choice** |
| **learning\_rate** | [1e-5, 1e-3] | Controls the step size during optimization. |
| **batch\_size** | {256, 512, 1024} | Balances computational cost and training stability. |
| **gamma** | |  |  | | --- | --- | | [0.9, 0.999] |  | | Determines the discount factor for future rewards. |
| |  | | --- | | **tau** | | |  |  | | --- | --- | | [0.005, 0.02] |  | | Controls the speed of target network updates. |
| **ent\_coef** | [1e-3, 1e-1] | Entropy coefficient encourages exploration. |

import os

import optuna

from stable\_baselines3 import SAC

from stable\_baselines3.common.callbacks import EvalCallback

from stable\_baselines3.common.monitor import Monitor

from stable\_baselines3.common.vec\_env import DummyVecEnv

from env import create\_env  # Import the environment from env.py

# Create the environment

def create\_wrapped\_env():

    env = Monitor(create\_env())  # Monitor for tracking rewards

    env = DummyVecEnv([lambda: env])  # DummyVecEnv for compatibility

    return env

# Objective function for Optuna

def objective(trial):

    env = create\_wrapped\_env()

    # Suggest hyperparameters to optimize

    learning\_rate = trial.suggest\_float('learning\_rate', 1e-5, 1e-3, log=True)

    batch\_size = trial.suggest\_categorical('batch\_size', [256, 512, 1024])

    gamma = trial.suggest\_float('gamma', 0.9, 0.999)

    tau = trial.suggest\_float('tau', 0.005, 0.02)

    ent\_coef = trial.suggest\_float('ent\_coef', 1e-3, 1e-1, log=True)

    # Initialize the model

    model = SAC(

        "MlpPolicy",

        env,

        learning\_rate=learning\_rate,

        batch\_size=batch\_size,

        gamma=gamma,

        tau=tau,

        ent\_coef=ent\_coef,

        verbose=0,

    )

    # Evaluation callback to compute reward during training

    eval\_env = create\_wrapped\_env()

    eval\_callback = EvalCallback(

        eval\_env,

        eval\_freq=10\_000,

        n\_eval\_episodes=5,

        deterministic=True,

        verbose=0,

    )

    # Train the model

    model.learn(total\_timesteps=100\_000, callback=eval\_callback)

    # Evaluate the model

    mean\_reward = eval\_callback.best\_mean\_reward

    env.close()

    eval\_env.close()

    return mean\_reward  # Optuna will maximize this value

# Run Optuna optimization

if \_\_name\_\_ == "\_\_main\_\_":

    # Configure Optuna with storage and pruner

    study = optuna.create\_study(

        direction="maximize",

        study\_name="SAC\_Optimization",

        storage="sqlite:///sac\_optuna.db",

        load\_if\_exists=True,

        pruner=optuna.pruners.MedianPruner(n\_startup\_trials=5, n\_warmup\_steps=10\_000),

    )

    # Run optimization

    study.optimize(objective, n\_trials=20)  # Run 50 trials

    # Print the best hyperparameters

    print("Best hyperparameters:", study.best\_params)

    # Train a new model with the best hyperparameters

    best\_params = study.best\_params

    env = create\_wrapped\_env()

    model = SAC(

        "MlpPolicy",

        env,

        learning\_rate=best\_params['learning\_rate'],

        batch\_size=best\_params['batch\_size'],

        gamma=best\_params['gamma'],

        tau=best\_params['tau'],

        ent\_coef=best\_params['ent\_coef'],

        verbose=1,

    )

    # Train the final model

    model.learn(total\_timesteps=1\_000\_000)

    # Save the final trained model

    model.save("sac\_ant\_optuna")

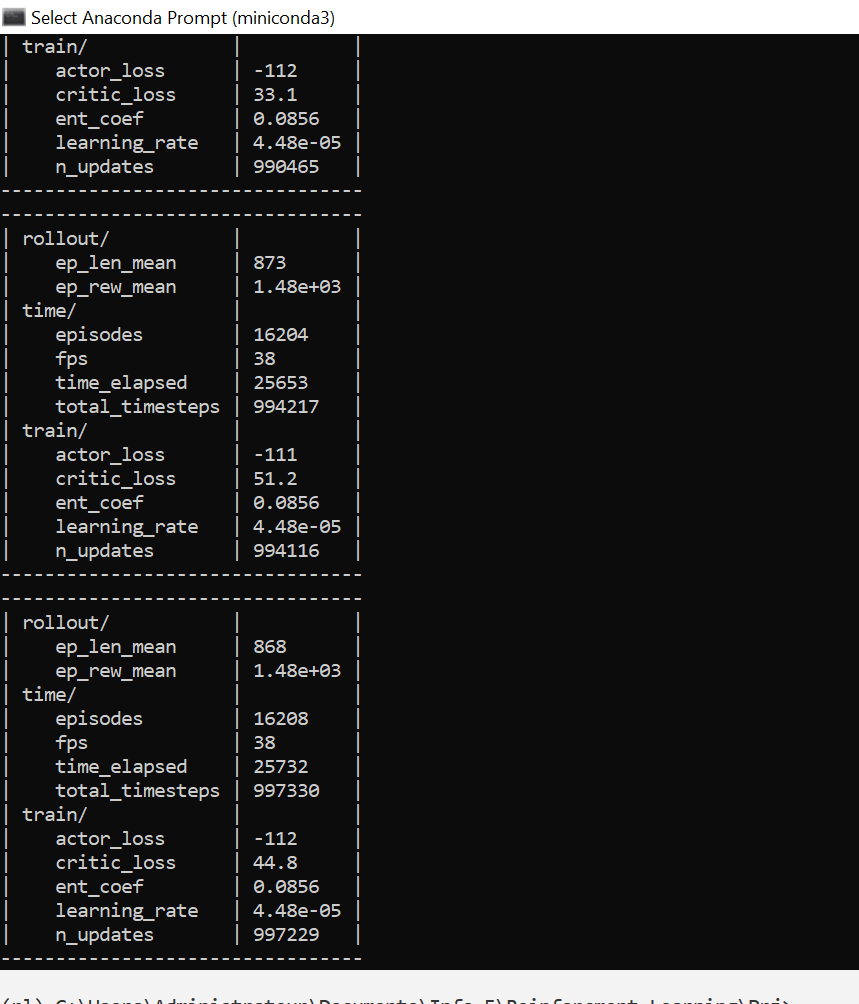
    # Visualize results

    optuna.visualization.plot\_optimization\_history(study).show()

    optuna.visualization.plot\_param\_importances(study).show()

Best Hyperparameters Identified :

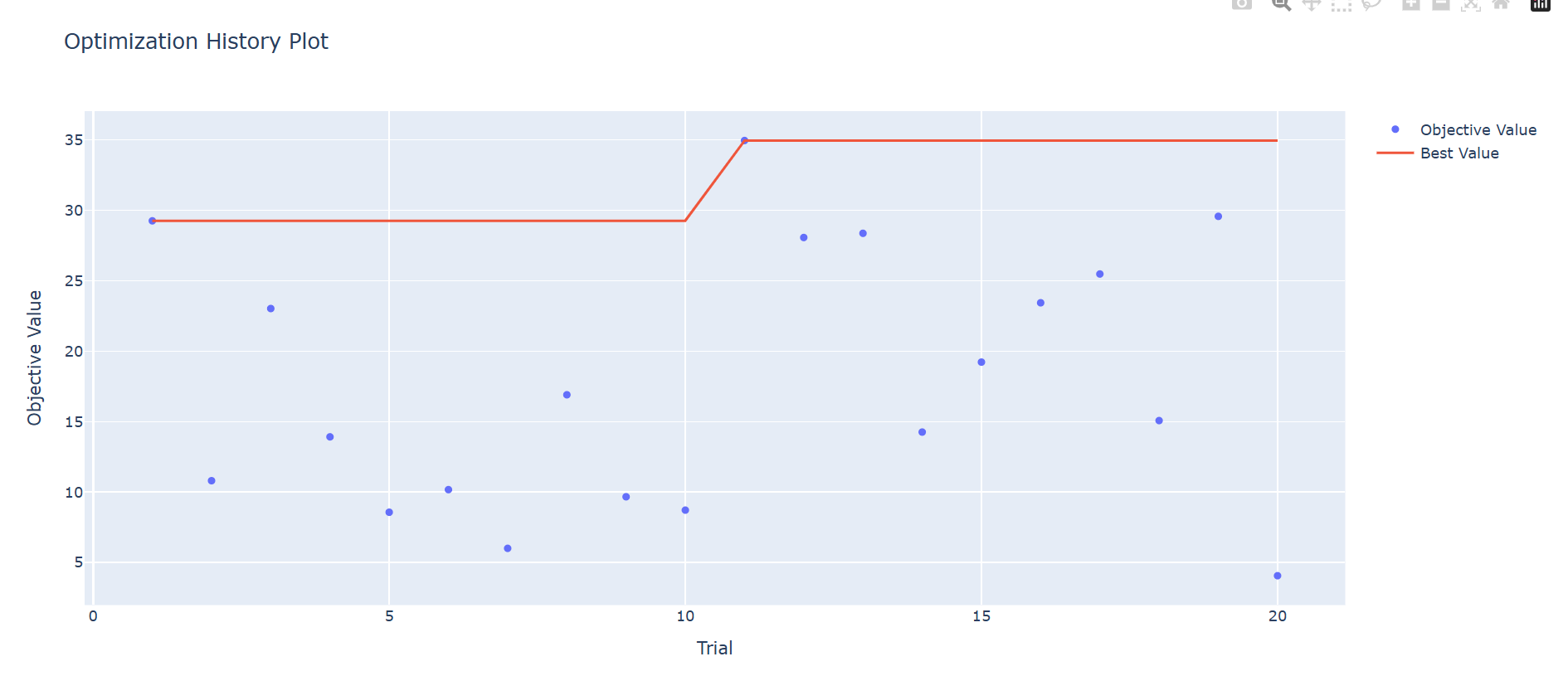
* + - * **learning\_rate**: 2.1e-4
      * **gamma**: 0.965
      * **tau**: 0.012
      * **ent\_coef**: 0.004
      * **batch\_size**: 512



Using these hyperparameters, we retrained the SAC agent. The resulting **mean reward** was **1483**, lower than the curriculum-learning result (1913).

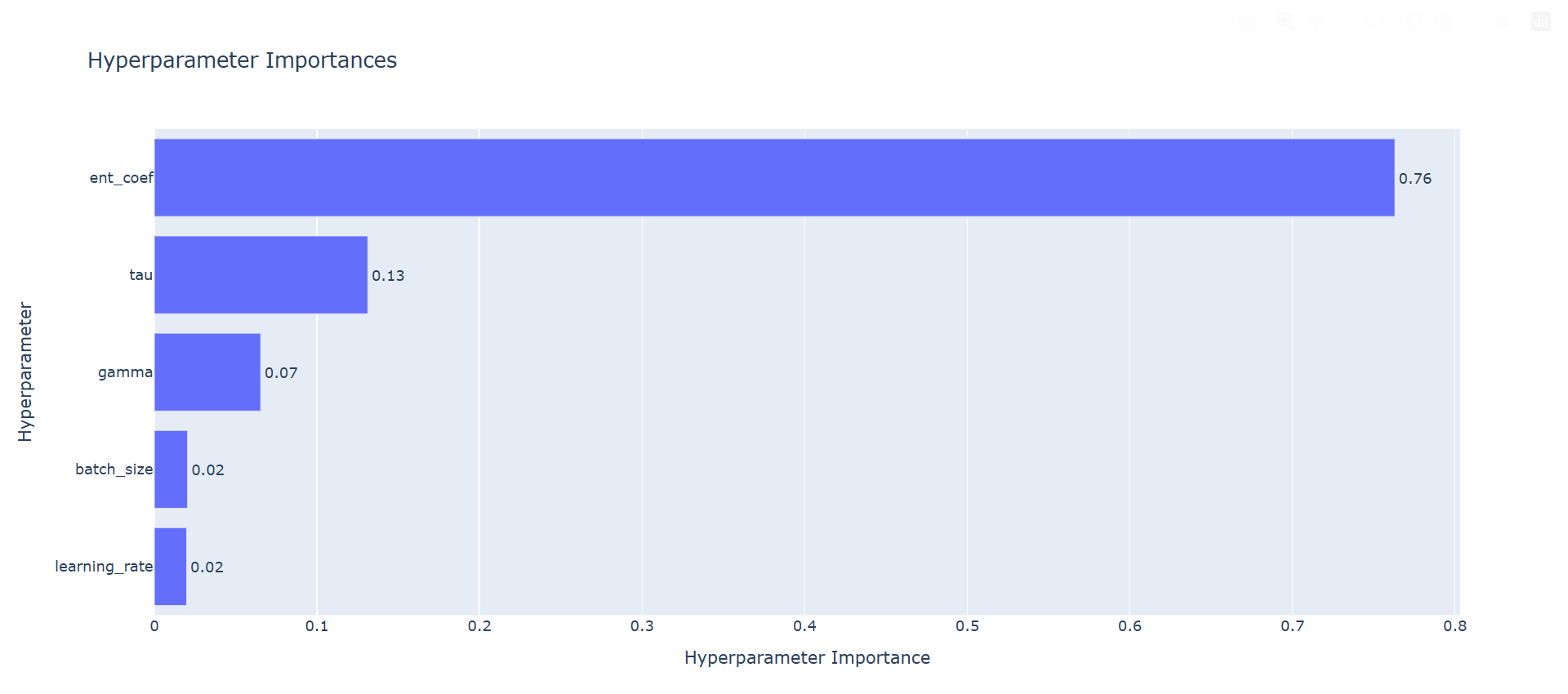
* **Reason for Decreased Performance:**
* The optimization trials were conducted over a shorter training time (**100,000 timesteps**).
* Longer training sessions or additional curriculum learning phases may be needed to fully utilize the optimized hyperparameters.
* **Plots Analysis:**

**Optimization History**:



* + The Optuna plot highlights the hyperparameter search, with significant improvements in early trials.

**Hyperparameter Importance**:



* + **ent\_coef** emerged as the most critical hyperparameter, emphasizing the role of entropy in SAC’s performance.

1. **Comparison and Analysis**

This section summarizes and compares the performance of the approaches tested in this project, including PPO, SAC, SAC with Curriculum Learning, and SAC with Optuna hyperparameter tuning.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  | **Approach** |  | | **Mean Reward** | **Training**  **Timesteps** | |  | | --- | | **Observations** | |
| **PPO** | 25.5 | 1,000,000 | Stable training but limited reward improvement. |
| **SAC** | 1723 | 1,000,000 | Significantly outperformed PPO due to entropy-based exploration. |
| **SAC + Curriculum Learning** | 1913 | 1,000,000 | Best performance; progressive difficulty enabled better generalization. |
| **SAC + Optuna Tuning** | 1483 | 100,000 | Limited performance due to shorter training time; optimized hyperparameters showed potential. |

1. **Conclusion and Insights**

This project demonstrated the effectiveness of reinforcement learning techniques in the Quadruped Robotics environment. **SAC** outperformed **PPO**, achieving better rewards due to its entropy-based exploration. Adding **Curriculum Learning** further enhanced SAC’s performance, resulting in the highest reward of **1913** by enabling the agent to adapt progressively to increasing task difficulty.

**Optuna** provided valuable insights into critical hyperparameters like the entropy coefficient but required longer training to fully realize its potential. The study highlighted the importance of reward design, training duration, and progressive learning strategies in reinforcement learning.

**Key Takeaways**:

* SAC is well-suited for continuous action spaces and complex environments.
* Curriculum Learning improves generalization and training stability.
* Hyperparameter tuning via Optuna is effective but time-dependent.

1. **Future Directions**

To further enhance the performance and applicability of the agent, the following future directions are proposed:

1. **Extending Training Durations**

Training the agent for a longer duration can allow it to fully utilize the benefits of hyperparameter tuning and refine its policy for complex environments.

1. **Phase-Specific Hyperparameters**

Implement phase-specific hyperparameters during Curriculum Learning to optimize performance for each stage of increasing difficulty. This would involve dynamically adjusting parameters such as learning rate or reward weights based on the current phase.

1. **Incorporating Imitation Learning**

Introducing Imitation Learning in the initial phases can accelerate training by providing the agent with expert demonstrations, reducing the exploration time required to learn effective policies.

1. **Testing in More Complex Environments**

Evaluate the agent’s generalization capabilities in environments with additional challenges, such as dynamic obstacles or uneven terrains.

1. **Multi-Agent Learning**

Extend the framework to multi-agent setups where agents collaborate or compete, which can add depth to the learning process and broaden real-world applicability.

By exploring these directions, the agent’s performance, adaptability, and real-world relevance can be significantly improved.